Intel® Celeron™ Processor Specification Update

Release Date: March 2000

Order Number: 243748-023

The Intel® Celeron™ processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

CONTENTS

REVISION HISTORY

Date of Revision	Version	Description
April 1998	-001	This document is the first Specification Update for the Intel® Celeron™ processor.
May 1998	-002	Added Errata 24 through 28.
June 1998	-003	Updated S-spec Table. Updated Summary Table of Changes. Updated Erratum 2 and 26. Added Errata 29 and 30. Added Documentation Changes 7 through 12. Added Specification Clarification 6 and 7.
July 1998	-004	Updated S-spec Table. Added Documentation Changes 13 through 16. Added Specification Clarifications 7 through 12. Added Specification Change 1.
August 1998	-005	Updated Summary Table of Changes. Changed numbering in order to maintain consistency with other product Specification Updates. Updated Errata 6 and 38. Added Errata 56 through 59. Updated Specification Clarification 5.
September 1998	-006	Updated S-spec table. Updated Erratum 56. Added Errata 60 through 62.
October 1998	-007	Implemented new numbering nomenclature. Updated Errata C1 and C27. Added Errata C37 through C39. Added Specification Clarification C15. Added Specification Change C2.
November 1998	-008	Updated Erratum C23. Added Erratum C40. Updated Documentation Change C10. Added Documentation Changes C17 and C18. Added Specification Change C3.
December 1998	-009	Added the Intel Celeron Processor (PPGA) markings. Added the mB0 stepping to the Processor Identification Information table and the Table of Changes. Added Errata C41 and C42.
December 1998	-010	Updated Identification Information table
January 1999	-011	Added Erratum C3AP. Added Documentation Changes C19 and C20. Updated Processor Identification Information table.
February 1999	-012	Updated Processor Identification Information table.
March 1999	-013	Updated Processor Markings, Summary Table of Changes, Documentation Changes, Specification Clarifications, and Specification Changes sections. Added Specification Change C1.
May 1999	-014	Updated the Processor Identification Information table. Added Erratum C43.
June 1999	-015	Added Erratum C44. Added Documentation Change C1. Added Specification Clarifications C2 and C3. Added Specification Change C1.
July 1999	-016	Added Erratum C45.

REVISION HISTORY

Date of Revision	Version	Description
August 1999	-017	Added Documentation Change C2. Updated Preface paragraph. Updated Codes Used in Summary Table. Updated column heading in Errata, Documentation Changes, Specification Clarifications and Specification Changes tables.
October 1999	-018	Added 'Brand Id' to *Identification Information* table. Updated *Processor Identification Information* Table. Added Errata C46.
November 1999	-019	Added Errata C47 and C48. Added Documentation Change C3.
December 1999	-020	Added Errata C49. Added Documentation Change C4. Added Specification Clarification C4.
January 2000	-021	Added Errata C50 and C51. Added Documentation Change C5.
February 2000	-022	Added Documentation Change C6. Updated Summary of Changes product letter codes.
March 2000	-023	Updated Erratum C47. Updated the CPUID/Stepping information in the Summary of Changes section.

PREFACE

This document is an update to the specifications contained in the following documents:

- *Pentium® II Processor Developer's Manual* (Order Number 243502)
- *P6 Family of Processors Hardware Developer's Manual* (Order Number 244001)
- *Intel® Celeron™ Processor* datasheet (Order Number 243658)
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3* (Order Numbers 243190, 243191, and 243192, respectively)

It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains S-Specs, Errata, Documentation Changes, Specification Clarifications and, Specification Changes.

Nomenclature

S-Spec Number is a five-digit code used to identify products. Products are differentiated by their unique characteristics, e.g., core speed, L2 cache size, package type, etc. as described in the processor identification information table. Care should be taken to read all notes associated with each S-Spec number.

Errata are design defects or errors. Errata may cause the Intel Celeron processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor must assume that all errata documented for that processor are present on all devices unless otherwise noted.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

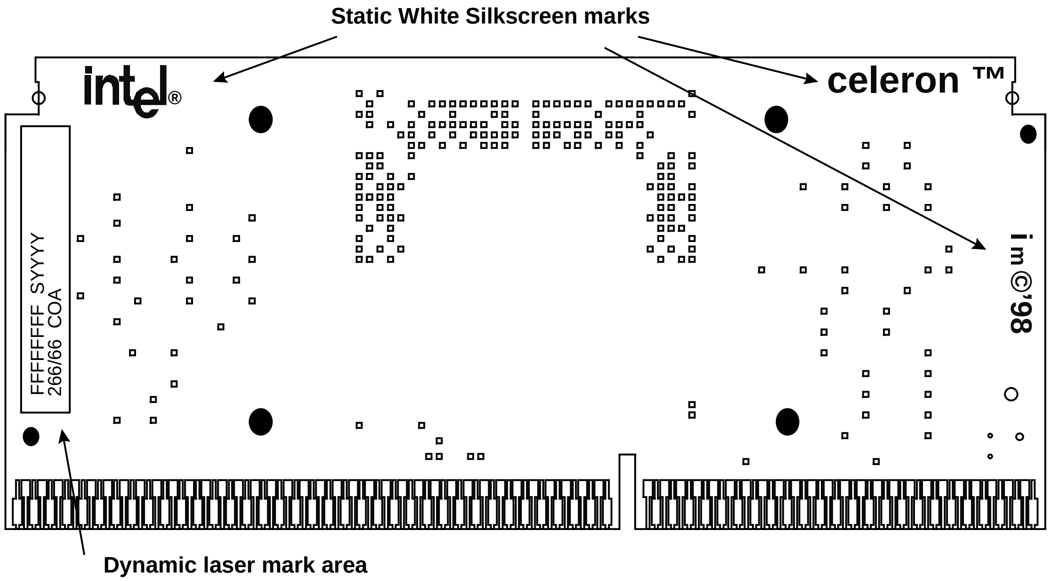
Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.

Specification Changes are modifications to the current published specifications for the Intel Celeron processor. These changes will be incorporated in the next release of the specifications.

Specification Update for the Intel® Celeron™ Processor

GENERAL INFORMATION

Intel® Celeron™ Processor and Boxed Intel® Celeron™ Processor Markings (S.E.P. Package)



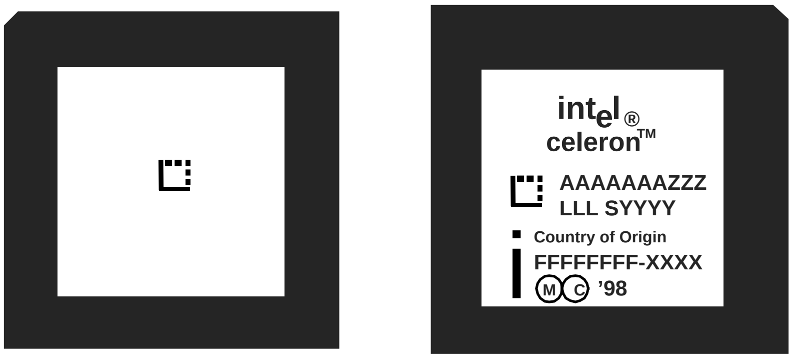
NOTES:

- SYYYY = S-spec Number.
- FFFFFFFF = FPO # (Test Lot Traceability #).

COA = Country of Assembly.

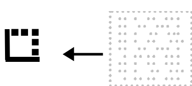
Intel® Celeron™ Processor and Boxed Intel® Celeron™ Processor Markings (PPGA Package)

Top **Bottom**



NOTES:

- AAAAAAA = Product Code
- ZZZ = Processor Speed (MHz)
- LLL = Integrated Level-Two Cache Size (in Kilobytes)
- SYYYY = S-Spec Number
- FFFFFFFF-XXXX = Assembly Lot Tracking Number



2-D Matrix Mark
Intel UCC#
Order Code (Product - speed)
S Number
Lot Number (date, factory)

IDENTIFICATION INFORMATION

Complete identification information of the Intel Celeron processor can be found in the *Intel Processor Identification and the CPUID Instruction* application note (Order Number 241618).

The Intel® Celeron™ processor can be identified by the following values:

Family[1]	Model[2]	Brand ID[3]
0110	0101	00h = Not Supported
0110	0110	00h = Not Supported

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after RESET, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after RESET, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.
3. The Brand ID corresponds to bits [7:0] of the EBX register after the CPUID instruction is executed with a 1 in the EAX register.

The Intel® Celeron™ processor's second level (L2) cache size can be determined by the following register contents:

0-Kbyte Unified L2 Cache[1]	40h
128-Kbyte Unified L2 Cache[1]	41h

NOTE:

For the Intel Celeron processor, the unified L2 cache size corresponds to the value in bits [3:0] of the EDX register after the CPUID instruction is executed with a 2 in the EAX register. Other Intel microprocessor models or families may move this information to other bit positions or otherwise reformat the result returned by this instruction; generic code should parse the resulting token stream according to the definition of the CPUID instruction.

Intel® Celeron™ Processor Identification Information

S-Spec	Core Stepping	L2 Cache Size (Kbytes)	CPUID	Speed (MHz) Core/Bus	Package and Revision	Notes
SL2SY	dA0	0	0650h	266/66	SEPP Rev. 1	
SL2YN	dA0	0	0650h	266/66	SEPP Rev. 1	1
SL2YP	dA0	0	0650h	300/66	SEPP Rev. 1	
SL2Z7	dA0	0	0650h	300/66	SEPP Rev. 1	1
SL2TR	dA1	0	0651h	266/66	SEPP Rev. 1	
SL2QG	dA1	0	0651h	266/66	SEPP Rev. 1	1
SL2X8	dA1	0	0651h	300/66	SEPP Rev. 1	
SL2Y2	dA1	0	0651h	300/66	SEPP Rev. 1	1
SL2Y3	dB0	0	0652h	266/66	SEPP Rev. 1	1
SL2Y4	dB0	0	0652h	300/66	SEPP Rev. 1	1
SL2WM	mA0	128	0660h	300A/66	SEPP Rev. 1	3
SL32A	mA0	128	0660h	300A/66	SEPP Rev. 1	1
SL2WN	mA0	128	0660h	333/66	SEPP Rev. 1	3
SL32B	mA0	128	0660h	333/66	SEPP Rev. 1	1
SL376	mA0	128	0660h	366/66	SEPP Rev. 1	
SL37Q	mA0	128	0660h	366/66	SEPP Rev. 1	1
SL39Z	mA0	128	0660h	400/66	SEPP Rev. 1	
SL37V	mA0	128	0660h	400/66	SEPP Rev. 1	1
SL3BC	mA0	128	0660h	433/66	SEPP Rev. 1	
SL35Q	mB0	128	0665h	300A/66	PPGA	2
SL36A	mB0	128	0665h	300A/66	PPGA	
SL35R	mB0	128	0665h	333/66	PPGA	2
SL36B	mB0	128	0665h	333/66	PPGA	
SL36C	mB0	128	0665h	366/66	PPGA	
SL35S	mB0	128	0665h	366/66	PPGA	2
SL3A2	mB0	128	0665h	400/66	PPGA	
SL37X	mB0	128	0665h	400/66	PPGA	2
SL3BA	mB0	128	0665h	433/66	PPGA	
SL3BS	mB0	128	0665h	433/66	PPGA	2

Intel® Celeron™ Processor Identification Information

S-Spec	Core Stepping	L2 Cache Size (Kbytes)	CPUID	Speed (MHz) Core/Bus	Package and Revision	Notes
SL3EH	mB0	128	0665h	466/66	PPGA	
SL3FL	mB0	128	0665h	466/66	PPGA	2
SL3FY	mB0	128	0665h	500/66	PPGA	
SL3LQ	mB0	128	0665h	500/66	PPGA	2
SL3FZ	mB0	128	0665h	533/66	PPGA	
SL3PZ	mB0	128	0665h	533/66	PPGA	2

NOTES:

1. This is a boxed Intel Celeron processor with an attached fan heatsink.
2. This is a boxed Intel Celeron processor with an unattached fan heatsink.
3. This part also ships as a boxed Intel Celeron processor with an attached fan heatsink.

SUMMARY OF CHANGES

The following table indicates the Errata, Documentation Changes, Specification Clarifications, or Specification Changes that apply to Intel Celeron processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Erratum, Documentation Change, Specification Clarification, or Specification Change applies to the given processor stepping.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
Doc:	Intel intends to update the appropriate documentation in a future revision.
PKG:	This column refers to errata on the Pentium® II processor substrate.
AP:	APIC related erratum.
Shaded:	This item is either new or modified from the previous version of the document.

Each Specification Update item is prefixed with a capital letter to distinguish the product. The key below details the letters that are used in Intel's microprocessor Specification Updates:

A = Intel® Pentium® II processor

B = Intel® Mobile Pentium® II processor

C = Intel® Celeron™ processor

D = Intel® Pentium® II Xeon™ processor

E = Intel® Pentium® III processor

G = Intel® Pentium® III Xeon™ processor

H = Intel® Mobile Celeron™ processor at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz

K = Intel® Mobile Pentium® III processor

M = Intel® Mobile Celeron™ processor at 500 MHz, 450 MHz, and 400A MHz

The Specification Updates for the Pentium® processor, Pentium® Pro processor, and other Intel products do not use this convention.

Summary of Errata

NO.	dA0	dA1	mA0	mB0	PKG	Plans	ERRATA
C1	X	X	X	X		NoFix	FP Data Operand Pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
C2	X	X	X	X		NoFix	Differences exist in debug exception reporting
C3	X	X	X	X		NoFix	Code fetch matching disabled debug register may cause debug exception
C4	X	X	X	X		NoFix	FP inexact-result exception flag may not be set
C5	X	X	X	X		NoFix	BTM for SMI will contain incorrect FROM EIP
C6	X	X	X	X		NoFix	I/O restart in SMM may fail after simultaneous MCE
C7	X	X	X	X		NoFix	Branch traps do not function if BTMs are also enabled
C8	X	X	X	X		NoFix	Machine check exception handler may not always execute successfully
C9	X	X	X	X		NoFix	LBER may be corrupted after some events
C10	X	X	X	X		NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
C11	X	X	X	X		Fix	Potential early deassertion of LOCK# during split-lock cycles
C12	X	X	X	X		NoFix	A20M# may be inverted after returning from SMM and Reset
C13	X	X				Fix	Reporting of floating-point exception may be delayed
C14	X	X	X	X		NoFix	Near CALL to ESP creates unexpected EIP address
C15	X	X				Fix	Built-in self test always gives nonzero result
C16	X	X	X	X		Fix	THERMTRIP# may not be asserted as specified
C17	X					Fixed	Cache state corruption in the presence of page A/D-bit setting and snoop traffic
C18	X					Fixed	Snoop cycle generates spurious machine check exception
C19	X	X				Fixed	MOVD/MOVQ instruction writes to memory prematurely
C20	X	X	X	X		NoFix	Memory type undefined for nonmemory operations
C21	X	X				Fixed	Bus protocol conflict with optimized chipsets

Summary of Errata

NO.	dA0	dA1	mA0	mB0	PKG	Plans	ERRATA
C22	X	X	X	X		NoFix	FP Data Operand Pointer may not be zero after power on or Reset
C23	X	X	X	X		NoFix	MOVD following zeroing instruction can cause incorrect result
C24	X	X	X	X		NoFix	Premature execution of a load operation prior to exception handler invocation
C25	X	X	X	X		NoFix	Read portion of RMW instruction may execute twice
C26	X	X	X	X		Fix	Test pin must be high during power up
C27	X	X	X	X		Fix	Intervening writeback may occur during locked transaction
C28	X	X	X	X		NoFix	MC2_STATUS MSR has model-specific error code and machine check architecture error code reversed
C29	X	X	X	X		NoFix	MOV with debug register causes debug exception
C30	X	X	X	X		NoFix	Upper four PAT entries not usable with Mode B or Mode C paging
C31	X	X				Fixed	Incorrect memory type may be used when MTRRs are disabled
C32	X	X	X			Fixed	Misprediction in program flow may cause unexpected instruction execution
C33	X	X	X	X		NoFix	Data Breakpoint Exception in a displacement relative near call may corrupt EIP
C34	X	X	X	X		NoFix	System bus ECC not functional with 2:1 ratio
C35	X	X	X			Fixed	Fault on REP CMPS/SCAS operation may cause incorrect EIP
C36	X	X	X	X		NoFix	RDMSR and WRMSR to invalid MSR address may not cause GP fault
C37	X	X	X	X		NoFix	SYSENTER/SYSEXIT instructions can implicitly load “null segment selector” to SS and CS registers
C38	X	X	X	X		NoFix	PRELOAD followed by EXTEST does not load boundary scan data
C39	X	X	X	X		NoFix	Far jump to new TSS with D-bit cleared may cause system hang
C40	X	X				Fixed	Incorrect chunk ordering may prevent execution of the machine check exception handler after BINIT#

Summary of Errata

NO.	dA0	dA1	mA0	mB0	PKG	Plans	ERRATA
C41	X	X	X			Fixed	UC write may be reordered around a cacheable write
C42	X	X	X	X		NoFix	Resume Flag may not be cleared after debug exception
C43			X	X		NoFix	Internal cache protocol violation may cause system hang
C44	X	X	X	X		NoFix	GP# fault on WRMSR to ROB_CR_BKUPTMPDR6
C45	X	X	X			NoFix	Machine Check Exception may occur due to improper line eviction in the IFU
C46	X	X	X	X		NoFix	Lower bits of SMRAM SMBASE register cannot be written with an ITP
C47	X	X	X	X		NoFix	Task switch may cause wrong PTE and PDE access bit to be set
C48	X	X	X	X		NoFix	Cross-modifying code operations on a jump instruction may cause a general protection fault
C49	X	X	X	X		NoFix	Deadlock may occur due to illegal-instruction/page-miss combination
C50	X	X	X	X		NoFix	FLUSH# assertion following STPCLK# may prevent CPU clocks from stopping
C51	X	X	X	X		NoFix	Floating-point exception condition may be deferred
C1AP	X	X	X	X		NoFix	APIC access to cacheable memory causes SHUTDOWN
C2AP	X	X	X	X		NoFix	Write to mask LVT (programmed as EXTINT) will not deassert outstanding interrupt
C3AP	X	X	X	X		NoFix	Misaligned locked access to APIC space results in hang

Summary of Documentation Changes

NO.	dA0	dA1	mA0	mB0	PKG	Plans	DOCUMENTATION CHANGES
C1	X	X	X	X		Doc	STPCLK# pin definition
C2	X	X	X	X		Doc	Invalidating caches and TLBs
C3	X	X	X	X		Doc	Handling of self-modifying and cross-modifying code
C4	X	X	X	X		Doc	Machine check architecture initialization of MCi_STATUS registers
C5	X	X	X	X		Doc	LOCK# signal prefix operands
C6	X	X	X	X		Doc	SMRAM state save map contains documentation errors

Summary of Specification Clarifications

NO.	dA0	dA1	mA0	mB0	PKG	Plans	SPECIFICATION CLARIFICATIONS
C1	X	X	X	X		Doc	PWRGOOD inactive pulse width
C2	X	X	X	X		Doc	Floating-point opcode clarification
C3	X	X	X	X		Doc	MTRR initialization clarification
C4	X	X	X	X		Doc	Non-AGTL+ output low current clarification

Summary of Specification Changes

NO.	dA0	dA1	mA0	mB0	PKG	Plans	SPECIFICATION CHANGES
C1	X	X	X	X		Doc	RESET# pin definition

ERRATA

C1. FP Data Operand Pointer May Be Incorrectly Calculated After FP Access Which Wraps 64-Kbyte Boundary in 16-Bit Code

Problem: The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved, the value contained in the FP Data Operand Pointer may be incorrect.

Implication: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load or store which wraps the 64-Kbyte boundary is executed.
- The operating system performs a floating-point environment store (FSAVE/FNSAVE/FSTENV/FNSTENV) after the above memory access.
- The operating system uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.

Workaround: If the FP Data Operand Pointer is used in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C2. Differences Exist in Debug Exception Reporting

Problem: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the Intel Celeron processor, as described below:

Case 1: The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Intel Celeron processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

Case 2: In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which crosses a 4-Kbyte page boundary, the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information across such a page split.

Case 3: If they occur after a MOVSS or POPSS instruction, the INT *n*, INTO, and INT3 instructions zero the DR6.Bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

Case 4: If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

Case 5: When an instruction which accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.

Implication: When debugging or when developing debuggers for an Intel Celeron processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4.

Workaround: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. No workaround has been identified for cases 4 or 5.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C3. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

Problem: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are *disabled* (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

Implication: While debugging software, extraneous instruction-breakpoint faults may be encountered if breakpoint registers are not cleared when they are disabled. Debug software which does not implement a code breakpoint handler will fail, if this occurs. If a handler is present, the fault will be serviced. Mixing data and code may exacerbate this problem by allowing disabled data breakpoint registers to break on an instruction fetch.

Workaround: The debug handler should clear breakpoint registers before they become disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C4. FP Inexact-Result Exception Flag May Not Be Set

Problem: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int
- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

Implication: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a “sticky” bit, i.e., once set by an inexact-result condition, it remains set until cleared by software.

Workaround: This condition can be avoided by inserting two NOP instructions between the two floating-point instructions.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C5. BTM for SMI Will Contain Incorrect FROM EIP

Problem: A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

Implication: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C6. I/O Restart in SMM May Fail After Simultaneous MCE

Problem: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the Intel Celeron processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

Implication: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and shutdown of the processor.

Workaround: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C7. Branch Traps Do Not Function If BTMs Are Also Enabled

Problem: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.

Implication: The branch traps and branch trace message debugging features cannot be used together.

Workaround: If branch trap functionality is desired, BTMs must be disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C8. Machine Check Exception Handler May Not Always Execute Successfully

Problem: An asynchronous machine check exception (MCE), such as a BINIT# event, which occurs during an access that splits a 4-Kbyte page boundary may leave some internal registers in an indeterminate state. Thus, MCE handler code may not always run successfully if an asynchronous MCE has occurred previously.

Implication: An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the processor to enter shutdown. Therefore, leaving MCEs disabled may not improve overall system behavior.

Workaround: No workaround which would guarantee successful MCE handler execution under this condition has been identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C9. LBER May Be Corrupted After Some Events

Problem: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the re-initialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.

Implication: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C10. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

Problem: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

Implication: Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C11. Potential Early Deassertion of LOCK# During Split-Lock Cycles

Problem: During a split-lock cycle there are four bus transactions: 1st ADS# (a partial read), 2nd ADS# (a partial read), 3rd ADS# (a partial write), and the 4th ADS# (a partial write). Due to this erratum, LOCK# may deassert one clock after the 4th ADS# of the split-lock cycle instead of after the 4th RS# assertion corresponding to the 4th ADS# has been sampled. The following sequence of events are required for this erratum to occur:

1. A lock cycle occurs (split or nonsplit).
2. Five more bus transactions (assertion of ADS#) occur.
3. A split-lock cycle occurs and BNR# toggles after the 3rd ADS# (partial write) of the split-lock cycle. This in turn delays the assertion of the 4th ADS# of the split-lock cycle. BNR# toggling at this time could most likely happen when the bus is set for an IOQ depth of 2.

When all of these events occur, LOCK# will be deasserted in the next clock after the 4th ADS# of the split-lock cycle.

Implication: This may affect chipset logic which monitors the behavior of LOCK# deassertion.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C12. A20M# May Be Inverted After Returning From SMM and Reset

Problem: This erratum is seen when software causes the following events to occur:

1. The assertion of A20M# in real address mode.
2. After entering the 1-Mbyte address wrap-around mode caused by the assertion of A20M#, there is an assertion of SMI# intended to cause a Reset or remove power to the processor. Once in the SMM handler, software saves the SMM state save map to an area of nonvolatile memory from which it can be restored at some point in the future. Then software asserts RESET# or removes power to the processor.
3. After exiting Reset or completion of power-on, software asserts SMI# again. Once in the SMM handler, it then retrieves the old SMM state save map which was saved in event 2 above and copies it into the current SMM state save map. Software then asserts A20M# and executes the RSM instruction. After exiting the SMM handler, the polarity of A20M# is inverted.

Implication: If this erratum occurs, A20M# will behave with a polarity opposite from what is expected (i.e., the 1-Mbyte address wrap-around mode is enabled when A20M# is deasserted, and does not occur when A20M# is asserted).

Workaround: Software should save the A20M# signal state in nonvolatile memory before an assertion of RESET# or a power down condition. After coming out of Reset or at power on, SMI# should be asserted again. During the restoration of the old SMM state save map described in event 3 above, the entire map should be restored, except for bit 5 of the byte at offset 7F18h. This bit should retain the value assigned to it when the SMM state save map was created in event 3. The SMM handler should then restore the original value of the A20M# signal.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C13. Reporting of Floating-Point Exception May Be Delayed

Problem: The Intel Celeron processor normally reports a floating-point exception for an instruction when the next floating-point or MMX™ technology instruction is executed. The assertion of FERR# and/or the INT 16 interrupt corresponding to the exception may be delayed until the floating-point or MMX technology instruction *after* the one which is expected to trigger the exception, if the following conditions are met:

1. A floating-point instruction causes an exception.
2. Before another floating-point or MMX technology instruction, any one of the following occurs:
   - A subsequent data access occurs to a page which has not been marked as accessed
   - Data is referenced which crosses a page boundary
   - A possible page-fault condition is detected which, when resolved, completes without faulting
3. The instruction causing event 2 above is followed by a MOVQ or MOVD store instruction.

Implication: This erratum only affects software which operates with floating-point exceptions unmasked. Software which requires floating-point exceptions to be visible on the next floating-point or MMX technology instruction, and which uses floating-point calculations on data which is then used for MMX technology instructions, may see a delay in the reporting of a floating-point instruction exception in some cases. Note that mixing floating-point and MMX technology instructions in this way is not recommended.

Workaround: Inserting a WAIT or FWAIT instruction (or reading the floating-point status register) between the floating-point instruction and the MOVQ or MOVD instruction will give the expected results. This is already the recommended practice for software.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C14. Near CALL to ESP Creates Unexpected EIP Address

Problem: As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP *after* the return value is pushed on the stack, not the value in the ESP register *before* the instruction executed.

Implication: Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

Workaround: If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an *indirect* call using ESP (e.g., CALL [ESP]). The saved version of ESP should be popped off the stack after the call returns.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C15. Built-in Self Test Always Gives Nonzero Result

Problem: The Built-in Self Test (BIST) of the Intel Celeron processor does not give a zero result to indicate a passing test. Regardless of pass or fail status, bit 6 of the BIST result in the EAX register after running BIST is set.

Implication: Software which relies on a zero result to indicate a passing BIST will indicate BIST failure.

Workaround: Mask bit 6 of the BIST result register when analyzing BIST results.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C16. THERMTRIP# May Not Be Asserted as Specified

Problem: THERMTRIP# is a signal on the Intel Celeron processor which is asserted when the core reaches a critical temperature during operation as detailed in the processor specification. The Intel Celeron processor may not assert THERMTRIP# until a much higher temperature than the one specified is reached.

Implication: The THERMTRIP# feature is not functional on the Intel Celeron processor. Note that this erratum can only occur when the processor is running with a T_{PLATE} temperature over the maximum specification of 75° C.

Workaround: Avoid operation of the Intel Celeron processor outside of the thermal specifications defined by the processor specifications.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C17. Cache State Corruption in the Presence of Page A/D-bit Setting and Snoop Traffic

Problem: If an operating system uses the Page Access and/or Dirty bit feature implemented in the Intel architecture and there is a significant amount of snoop traffic on the bus, while the processor is setting the Access and/or Dirty bit the processor may inappropriately change a single L1 cache line to the modified state.

Implication: The occurrence of this erratum may result in cache incoherency, which may cause parity errors, data corruption (with no parity error), unexpected application or operating system termination, or system hangs.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C18. Snoop Cycle Generates Spurious Machine Check Exception

Problem: The processor may incorrectly generate a Machine Check Exception (MCE) when it processes a snoop access that does not hit the L1 data cache. Due to an internal logic error, this type of snoop cycle may still check data parity on undriven data lines. The processor generates a spurious machine check exception as a result of this unnecessary parity check.

Implication: A spurious machine check exception may result in an unexpected system halt if Machine Check Exception reporting is enabled in the operating system.

Workaround: It is possible for BIOS code to contain a workaround for this erratum. This workaround would fix the erratum, however, the reporting of the data parity error will continue.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C19. MOVD/MOVQ Instruction Writes to Memory Prematurely

Problem: When an instruction encounters a fault, the faulting instruction should not modify any CPU or system state. However, when the MMX™ technology store instructions MOVD and MOVQ encounter any of the following events, it is possible for the store to be committed to memory even though it should be canceled:

1. If CR0.EM = 1 (Emulation bit), then the store could happen prior to the triggered invalid opcode exception.
2. If the floating-point Top-of-Stack (FP TOS) is not zero, then the store could happen prior to executing the processor assist routine that sets the FP TOS to zero.
3. If there is an unmasked floating-point exception pending, then the store could happen prior to the triggered unmasked floating-point exception.
4. If CR0.TS = 1 (Task Switched bit), then the store could happen prior to the triggered Device Not Available (DNA) exception.

If the MOVD/MOVQ instruction is restarted after handling any of the above events, then the store will be performed again, overwriting with the expected data. The instruction will not be restarted after event 1. The instruction will definitely be restarted after events 2 and 4. The instruction may or may not be restarted after event 3, depending on the specific exception handler.

Implication: This erratum causes unpredictable behavior in an application if MOVD/MOVQ instructions are used to manipulate semaphores for multiprocessor synchronization, or if these MMX instructions are used to write to uncacheable memory or memory mapped I/O that has side effects, e.g., graphics devices. This erratum is completely transparent to all applications that do not have these characteristics. When each of the above conditions are analyzed:

1. Setting the CR0.EM bit forces all floating-point/MMX instructions to be handled by software emulation. The MOVD/MOVQ instruction, which is an MMX instruction, would be considered an invalid instruction. Operating systems typically terminates the application after getting the expected invalid opcode fault.
2. The FP TOS not equal to 0 case only occurs when the MOVD/MOVQ store is the first MMX instruction in an MMX technology routine and the previous floating-point routine did not clean up the floating-point states properly when it exited. Floating-point routines commonly leave TOS to 0 prior to exiting. For a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.

3. The unmasked floating-point exception case only occurs if the store is the first MMX technology instruction in an MMX technology routine and the previous floating-point routine exited with an unmasked floating-point exception pending. Again, for a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.

Device Not Available (DNA) exceptions occur naturally when a task switch is made between two tasks that use either floating-point instructions and/or MMX instructions. For this erratum, in the event of the DNA exception, data from the prior task may be temporarily stored to the present task's program state.

Workaround: Do not use MMX instructions to manipulate semaphores for multiprocessor synchronization. Do not use MOVD/MOVQ instructions to write directly to I/O devices if doing so triggers user visible side effects. An OS can prevent old data from being stored to a new task's program state by cleansing the FPU explicitly after every task switch. Follow Intel's recommended programming paradigms in the *Intel Architecture Developer's Optimization Manual* for writing MMX technology programs. Specifically, do not mix floating-point and MMX instructions. When transitioning to new a MMX technology routine, begin with an instruction that does not depend on the prior state of either the MMX technology registers or the floating-point registers, such as a load or PXOR mm0, mm0. Be sure that the FP TOS is clear before using MMX instructions.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C20. Memory Type Undefined for Nonmemory Operations

Problem: The Memory Type field for nonmemory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for nonmemory operations logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.

Implication: Bus agents may decode a non-UC memory type for nonmemory bus transactions.

Workaround: Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C21. Bus Protocol Conflict With Optimized Chipsets

Problem: A "dead" turnaround cycle with no agent driving the address, address parity, request command, or request parity signals must occur between the processor driving these signals and the chipset driving them after asserting BPRI#. The Intel Celeron processor does not follow this protocol. Thus, if a system uses a chipset or third party agent which optimizes its arbitration latency (reducing it to 2 clocks when it observes an active (low) ADS# signal and an inactive (high) LOCK# signal on the same clock that BPRI# is asserted (driven low)), the Intel Celeron processor may cause bus contention during an unlocked bus exchange.

Implication: This violation of the bus exchange protocol when using a reduced arbitration latency may cause a system-level setup timing violation on the address, address parity, request command, or request parity signals on the system bus. This may result in a system hang or assertion of the AERR# signal, causing an attempted corrective action or shutdown of the system, as the system hardware and software dictate. The possibility of failure due to the contention caused by this erratum may be increased due to the processor's internal active pull-up of these signals on the clock after the signals are no longer being driven by the processor.

Workaround: If the chipset and third party agents used with the Intel Celeron processor do not optimize their arbitration latency as described above, no action is required. For the 66 MHz Intel Celeron processor, no action is required.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C22. FP Data Operand Pointer May Not Be Zero After Power On or Reset

Problem: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.

Implication: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting in incorrect behavior of the software.

Workaround: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C23. MOVD Following Zeroing Instruction Can Cause Incorrect Result

Problem: An incorrect result may be calculated after the following circumstances occur:

1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,
2. A value is moved with sign extension into the same register's lower 16 bits; or a signed integer multiply is performed to the same register's lower 16 bits,
3. This register is then copied to an MMX™ technology register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX technology register. Only the MMX technology register is affected by this erratum.

The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.

1. XOR EAX, EAX
or SUB EAX, EAX
2. MOVSX AX, BL
or MOVSX AX, byte ptr <memory address> or MOVSX AX, BX
or MOVSX AX, word ptr <memory address> or IMUL BL (AX implicit, opcode F6 /5)
or IMUL byte ptr <memory address> (AX implicit, opcode F6 /5) or IMUL AX, BX (opcode 0F AF /r)
or IMUL AX, word ptr <memory address> (opcode 0F AF /r) or IMUL AX, BX, 16 (opcode 6B /r ib)
or IMUL AX, word ptr <memory address>, 16 (opcode 6B /r ib) or IMUL AX, 8 (opcode 6B /r ib)
or IMUL AX, BX, 1024 (opcode 69 /r iw)
or IMUL AX, word ptr <memory address>, 1024 (opcode 69 /r iw) or IMUL AX, 1024 (opcode 69 /r iw)
or CBW
3. MOVD MM0, EAX

Note that the values for immediate byte/words are merely representative (i.e., 8, 16, 1024) and that any value in the range for the size may be affected. Also, note that this erratum may occur with "EAX" replaced with any 32-bit general purpose register, and "AX" with the corresponding 16-bit version of that replacement. "BL" or "BX" can be replaced with any 8-bit or 16-bit general purpose register. The CBW and IMUL (opcode F6 /5) instructions are specific to the EAX register only.

In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the four types of the MOVSX or IMUL instructions and the CBW instruction modify only bits 15:8 of EAX by sign extending the lower 8 bits of EAX, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX, IMUL or CBW instruction is negative, i.e., bit 15 of AX is a 1.

When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX, IMUL or CBW instruction.

Implication: The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX technology-enabled application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.

Workaround: There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX technology for operating on multiple variables. This would result in higher performance as well.
2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/IMUL/CBW instruction and the MOVD instruction as in the example below:

```
XOR EAX, EAX (or SUB EAX, EAX)
MOVSX AX, BL (or other MOVSX, other IMUL or CBW instruction)
*MOV EAX, EAX
MOVD MM0, EAX
```

*Note: MOV EAX, EAX is used here as it is fairly generic. Again, EAX can be any 32-bit register.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C24. Premature Execution of a Load Operation Prior to Exception Handler Invocation

Problem: This erratum can occur with any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation
2. If a waiting floating-point instruction or MMX™ instruction that performs a memory load has a floating-point exception pending
3. If an MMX instruction that performs a memory load and has either CR0.EM =1 (Emulation bit set), or a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending

If any of the above circumstances occur, it is possible that the load portion of the instruction will have executed before the exception handler is entered.

Implication: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side effect, restarting the instruction may cause unexpected system behavior due to the repetition of the side effect.

Workaround: Code which performs loads from memory that has side-effects can effectively workaround this behavior by using simple integer-based load instructions when accessing side-effect memory and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading from side-effect memory.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C25. Read Portion of RMW Instruction May Execute Twice

Problem: When the Intel Celeron processor executes a read-modify-write (RMW) arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes.

If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.

Implication: This erratum has no effect if the memory targeted for the RMW instruction has no side-effects. If, however, the load targets a memory region that has side-effects, multiple occurrences of the initial load may lead to unpredictable system behavior.

Workaround: Hardware and software developers who write device drivers for custom hardware that may have a side-effect style of design should use simple loads and simple stores to transfer data to and from the device. Then the memory location will simply be read twice with no additional implications.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C26. Test Pin Must Be High During Power Up

Problem: The Intel Celeron processor uses the PWRGOOD signal to ensure that no voltage sequencing issues arise; no pin assertions should cause the processor to change its behavior until this signal is asserted, when all power supplies and clocks to the processor are valid and stable. However, if the TESTHI signal is at a low voltage level when the core power supply comes up, it will cause the processor to enter an invalid test state.

Implication: If this erratum occurs, the system may boot normally however, L2 cache may not be initialized.

Workaround: Ensure that the 2.5 V ($V_{CC_{2.5}}$) power supply ramps at or before the 2.0 V ($V_{CC_{CORE}}$) power plane. If 2.5 V ramps after core, pull up TESTHI to 2.5 V ($V_{CC_{2.5}}$) with a 100K Ohm resistor. The internal pull-up will keep the signal from being asserted during power up. For new motherboard designs, it is recommended that TESTHI be pulled up to 2.0 V ($V_{CC_{CORE}}$) using a 1K-10K Ohm resistor.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C27. Intervening Writeback May Occur During Locked Transaction

Problem: During a transaction which has the LOCK# signal asserted (i.e., a locked transaction), there is a potential for an explicit writeback caused by a previous transaction to complete while the bus is locked. The explicit writeback will only be issued by the processor which has locked the bus, and the lock signal will not be deasserted until the locked transaction completes, but the atomicity of a lock may be compromised by this erratum. Note that the explicit writeback is an expected cycle, and no memory ordering violations will occur. This erratum is, however, a violation of the bus lock protocol.

Implication: A chipset or third-party agent (TPA) which tracks bus transactions in such a way that locked transactions may only consist of a read-write or read-read-write-write locked sequence, with no transactions intervening, may lose synchronization of state due to the intervening explicit writeback. Systems using chipsets or TPAs which can accept the intervening transaction will not be affected.

Workaround: The bus tracking logic of all devices on the system bus should allow for the occurrence of an intervening transaction during a locked transaction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C28. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

Problem: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* documents that for the MC*i*_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code fields and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.

Implication: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.

Workaround: When decoding the MC2_STATUS MSR, reverse the two error fields.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C29. MOV With Debug Register Causes Debug Exception

Problem: When in V86 mode, if a MOV instruction is executed on debug registers, a general-protection exception (#GP) should be generated, as documented in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Section 15.2. However, in the case when the general detect enable flag (GD) bit is set, the observed behavior is that a debug exception (#DB) is generated instead.

Implication: With debug-register protection enabled (i.e., the GD bit set), when attempting to execute a MOV on debug registers in V86 mode, a debug exception will be generated instead of the expected general-protection fault.

Workaround: In general, operating systems do not set the GD bit when they are in V86 mode. The GD bit is generally set and used by debuggers. The debug exception handler should check that the exception did not occur in V86 mode before continuing. If the exception did occur in V86 mode, the exception may be directed to the general-protection exception handler.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C30. Upper Four PAT Entries Not Usable With Mode B or Mode C Paging

Problem: The Page Attribute Table (PAT) contains eight entries, which must all be initialized and considered when setting up memory types for the Intel Celeron processor. However, in Mode B or Mode C paging, the upper four entries do not function correctly for 4-Kbyte pages. Specifically, bit seven of page table entries that translate addresses to 4-Kbyte pages should be used as the upper bit of a three-bit index to determine the PAT entry that specifies the memory type for the page. When Mode B (CR4.PSE = 1) and/or Mode C (CR4.PAE) are enabled, the processor forces this bit to zero when determining the memory type regardless of the value in the page table entry. The upper four entries of the PAT function correctly for 2-Mbyte and 4-Mbyte large pages (specified by bit 12 of the page directory entry for those translations).

Implication: Only the lower four PAT entries are useful for 4-Kbyte translations when Mode B or C paging is used. In Mode A paging (4-Kbyte pages only), all eight entries may be used. All eight entries may be used for large pages in Mode B or C paging.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C31. Incorrect Memory Type May Be Used When MTRRs Are Disabled

Problem: If the Memory Type Range Registers (MTRRs) are disabled without setting the CR0.CD bit to disable caching, and the Page Attribute Table (PAT) entries are left in their default setting, which includes UC- memory type (PCD = 1, PWT = 0; see the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* for details), data for entries set to UC- will be cached as if the memory type were writeback (WB). Also, if the page tables are set to a memory type other than UC-, then the effective memory type used will be that specified by the page tables and PAT. Any regions of memory normally forced to UC by the MTRRs (such as the VGA video region) may now be incorrectly cached and speculatively accessed.

Even if the CR0.CD bit is correctly set when the MTRRs are disabled and the PAT is left in its default state, then retries and out of order retirement of UC accesses may occur, contrary to the strong ordering expected for these transactions.

Implication: The occurrence of this erratum may result in the use of incorrect data and unpredictable processor behavior when running with the MTRRs disabled. Interaction between the mouse, cursor, and VGA video display leading to video corruption may occur as a symptom of this erratum as well.

Workaround: Ensure that when the MTRRs are disabled, the CR0.CD bit is set to disable caching. This recommendation is described in *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. If it is necessary to disable the MTRRs, first clear the PAT register before setting the CR0.CD bit, flushing the caches, and disabling the MTRRs to ensure that UC memory type is always returned and strong ordering is maintained.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C32. Misprediction in Program Flow May Cause Unexpected Instruction Execution

Problem: To optimize performance through dynamic execution technology, the P6 architecture has the ability to predict program flow. In the event of a misprediction, the processor will normally clear the incorrect prediction, adjust the EIP to the correct location, and flush out any instructions it may have fetched from the misprediction. In circumstances where a branch misprediction occurs, the correct target of the branch has already been opportunistically fetched into the streaming buffers, and the L2 cycle caused by the evicted cache line is retried by the L2 cache, the processor may fail to flush out the retirement unit before the speculative program flow is committed to a permanent state.

Implication: The results of this erratum may range from no effect to unpredictable application or OS failure. Manifestations of this failure may result in:

- Unexpected values in EIP
- Faults or traps (e.g., page faults) on instructions that do not normally cause faults
- Faults in the middle of instructions
- Unexplained values in registers/memory at the correct EIP

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C33. Data Breakpoint Exception in a Displacement Relative Near Call May Corrupt EIP

Problem: If a data breakpoint is programmed at the memory location where the stack push of a near call is performed, the processor will update the stack and ESP appropriately, but may skip the code at the destination of the call. Hence, program execution will continue with the next instruction immediately following the call, instead of the target of the call.

Implication: The failure mechanism for this erratum is that the call would not be taken; therefore, instructions in the called subroutine would not be executed. As a result, any code relying on the execution of the subroutine will behave unpredictably.

Workaround: Do not program a data breakpoint exception on the stack where the push for the near call is performed.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C34. System Bus ECC Not Functional With 2:1 Ratio

Problem: If a processor is underclocked at a core frequency to system bus frequency ratio of 2:1 and system bus ECC is enabled, the system bus ECC detection and correction will negatively affect internal timing dependencies.

Implication: If system bus ECC is enabled, and the processor is underclocked at a 2:1 ratio, the system may behave unpredictably due to these timing dependencies.

Workaround: All bus agents that support system bus ECC must disable it when a 2:1 ratio is used.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C35. Fault on REP CMPS/SCAS Operation May Cause Incorrect EIP

Problem: If either a General Protection Fault, Alignment Check Fault or Machine Check Exception occur during the first iteration of a REP CMPS or a REP SCAS instruction, an incorrect EIP may be pushed onto the stack of the event handler if all the following conditions are true:

- The event occurs on the initial load performed by the instruction(s)
- The condition of the zero flag before the repeat instruction happens to be opposite of the repeat condition (i.e., REP/REPE/REPZ CMPS/SCAS with ZF = 0 or RENE/REPNZ CMPS/SCAS with ZF = 1)
- The faulting micro-op and a particular micro-op of the REP instruction are retired in the retirement unit in a specific sequence

The EIP will point to the instruction following the REP CMPS/SCAS instead of pointing to the faulting instruction.

Implication: The result of the incorrect EIP may range from no effect to unexpected application/OS behavior.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C36. RDMSR or WRMSR To Invalid MSR Address May Not Cause GP Fault

Problem: The RDMSR and WRMSR instructions allow reading or writing of MSRs (Model Specific Registers) based on the index number placed in ECX. The processor should reject access to any reserved or unimplemented MSRs by generating #GP(0). However, there are some invalid MSR addresses for which the processor will not generate #GP(0).

Implication: For RDMSR, undefined values will be read into EDX:EAX. For WRMSR, undefined processor behavior may result.

Workaround: Do not use invalid MSR addresses with RDMSR or WRMSR.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C37. SYSENTER/SYSEXIT Instructions Can Implicitly Load "Null Segment Selector" to SS and CS Registers

Problem: According to the processor specification, attempting to load a null segment selector into the CS and SS segment registers should generate a General Protection Fault (#GP). Although loading a null segment selector to the other segment registers is allowed, the processor will generate an exception when the segment register holding a null selector is used to access memory.

However, the SYSENTER instruction can implicitly load a null value to the SS segment selector. This can occur if the value in SYSENTER_CS_MSR is between FFF8h and FFFBh when the SYSENTER instruction is executed. This behavior is part of the SYSENTER/SYSEXIT instruction definition; the content of the SYSTEM_CS_MSR is always incremented by 8 before it is loaded into the SS. This operation will set the null bit in the segment selector if a null result is generated, but it does not generate a #GP on the SYSENTER instruction itself. An exception will be generated as expected when the SS register is used to access memory, however.

The SYSEXIT instruction will also exhibit this behavior for both CS and SS when executed with the value in SYSENTER_CS_MSR between FFF0h and FFF3h, or between FFE8h and FFEBh.

Implication: These instructions are intended for operating system use. If this erratum occurs (and the OS does not ensure that the processor never has a null segment selector in the SS or CS segment registers), the processor's behavior may become unpredictable, possibly resulting in system failure.

Workaround: Do not initialize the SYSTEM_CS_MSR with the values between FFF8h and FFFBh, FFF0h and FFF3h, or FFE8h and FFEBh before executing SYSENTER or SYSEXIT.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C38. PRELOAD Followed by EXTEST Does Not Load Boundary Scan Data

Problem: According to the IEEE 1149.1 Standard, the EXTEST instruction would use data "typically loaded onto the latched parallel outputs of boundary-scan shift-register stages using the SAMPLE/PRELOAD instruction prior to the selection of the EXTEST instruction." As a result of this erratum, this method cannot be used to load the data onto the outputs.

Implication: Using the PRELOAD instruction prior to the EXTEST instruction will not produce expected data after the completion of EXTEST.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C39. Far Jump to New TSS With D-bit Cleared May Cause System Hang

Problem: A task switch may be performed by executing a far jump through a task gate or to a new Task State Segment (TSS) directly. Normally, when such a jump to a new TSS occurs, the D-bit (which indicates that the page referenced by a Page Table Entry (PTE) has been modified) for the PTE which maps the location of the previous TSS will already be set and the processor will operate as expected. However, if the D-bit is clear at the time of the jump to the new TSS, the processor will hang.

Implication: If an OS is used which can clear the D-bit for system pages, and which jumps to a new TSS on a task switch, then a condition may occur which results in a system hang. Intel has not identified any commercial software which may encounter this condition; this erratum was discovered in a focused testing environment.

Workaround: Ensure that OS code does not clear the D-bit for system pages (including any pages that contain a task gate or TSS). Use task gates rather than jumping to a new TSS when performing a task switch.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C40. Incorrect Chunk Ordering May Prevent Execution of the Machine Check Exception Handler After BINIT#

Problem: If a catastrophic bus error is detected which results in a BINIT# assertion, and the BINIT# assertion is propagated to the processor's L2 cache at the same time that data is being sent to the processor, then the data may become corrupted in the processor's L1 cache.

Implication: Since BINIT# assertion is due to a catastrophic event on the bus, the corrupted data will not be used. However, it may prevent the processor from executing the Machine Check Exception (MCE) handler, causing the system to hang.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C41. UC Write May Be Reordered Around a Cacheable Write

Problem: After a write occurs to a UC (uncacheable) region of memory, there exists a small window of opportunity where a subsequent write transaction targeted for a UC memory region may be reordered in front of a write targeted to a region of cacheable memory. This erratum can only occur during the following sequence of bus transactions:

1. A write to memory mapped as UC occurs
2. A write to memory mapped as cacheable (WB or WT) which is present in Shared or Invalid state in the L2 cache occurs
3. During the bus snoop of the cacheable line, another store to UC memory occurs

Implication: If this erratum occurs, the second UC write will be observed on the bus prior to the Bus Invalidate Line (BIL) or Bus Read Invalidate Line (BRIL) transaction for the cacheable write. This presents a small window of opportunity for a fast bus-mastering I/O device which triggers an action based on the second UC write to arbitrate and gain ownership of the bus prior to the completion of the cacheable write, possibly retrieving stale data.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C42. Resume Flag May Not Be Cleared After Debug Exception

Problem: The Resume Flag (RF) is normally cleared by the processor after executing an instruction which causes a debug exception (#DB). In the process of determining whether the RF needs to be cleared after executing the instruction, the processor uses an internal register containing stale data. The stale data may unpredictably prevent the processor from clearing the RF.

Implication: If this erratum occurs, further debug exceptions will be disabled.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C43. Internal Cache Protocol Violation May Cause System Hang

Problem: An Intel Celeron processor based system may hang due to an internal cache protocol violation. During multiple transactions targeted at the same cacheline, there exists a small window of time such that the processor's internal timings align to create a livelock situation. The scenario, which results in the erratum, is summarized below:
Scenario:

1. A snoopable transaction is issued to address A. This snoopable transaction can be issued by the processor or the chipset.
2. The snoopable transaction hits a modified line in the processor's L1 data cache.
3. The processor issues two code fetches from the L2 cache before the snoopable transaction reaches the top of the In-Order Queue and before the snoopable transaction's modified L1 cache line containing address A is brought out on the system bus.

At the same time, a locked access to the L1 cache occurs.

Implication: An Intel Celeron processor may cause a system to hang if the above listed sequence of events occur. The probability of encountering this erratum increases with I/O queue depth greater than four.

Workaround: It is possible for the BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C44. GP# Fault on WRMSR to ROB_CR_BKUPTMPDR6

Problem: Writing a '1' to unimplemented bit(s) in the ROB_CR_BKUPTMPDR6 MSR (offset 1E0h) will result in a general protection fault (GP#).

Implication: The normal process used to write an MSR is to read the MSR using RDMSR, modify the bit(s) of interest, and then to write the MSR using WRMSR. Because of this erratum, this process may result in a GP# fault when used to modify the ROB_CR_BKUPTMPDR6 MSR.

Workaround: When writing to ROB_CR_BKUPTMPDR6 all unimplemented bits must be '0.' Implemented bits may be set as '0' or '1' as desired.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C45. Machine Check Exception May Occur Due to Improper Line Eviction in the IFU

Problem: The Intel Celeron processor is designed to signal an unrecoverable Machine Check Exception (MCE) as a consistency checking mechanism. Under a complex set of circumstances involving multiple speculative branches and memory accesses there exists a one cycle long window in which the processor may signal a MCE in the Instruction Fetch Unit (IFU) because instructions previously decoded have been evicted from the IFU. The one cycle long window is opened when an opportunistic fetch receives a partial hit on a previously executed but not as yet completed store resident in the store buffer. The resulting partial hit erroneously causes the eviction of a line from the IFU at a time when the processor is expecting the line to still be present. If the MCE for this particular IFU event is disabled, execution will continue normally.

Implication: Since the probability of this erratum occurring increases with the number of processors, the risk is lower on Intel Celeron processor-based systems as they do not have multi-processor support. If this erratum does occur, a machine check exception will result. Note systems that implement an operating system that does not enable the Machine Check Architecture will be completely unaffected by this erratum (e.g., Windows* 95 and Windows 98).

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C46. Lower Bits of SMRAM SMBASE Register Cannot Be Written With an ITP

Problem: The System Management Base (SMBASE) register (7EF8H) stores the starting address of the System Management RAM (SMRAM). This register is used by the processor when it is in System Management Mode (SMM), and its contents serve as the memory base for code execution and data storage. The 32-bit SMBASE register can normally be programmed to any value. When programmed with an In-Target Probe (ITP), however, any attempt to set the lower 11 bits of SMBASE to anything other than zeros via the WRMSR instruction will cause the attempted write to fail.

Implication: When set via an ITP, any attempt to relocate SMRAM space must be made with 2-Kbyte alignment.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C47. Task Switch May Cause Wrong PTE and PDE Access Bit to be Set

Problem: When an operating system executes a task switch via a Task State Segment (TSS), and the TSS is wholly or partially located within a clean page (A and D bits clear) and the GDT entry for the new TSS is either misaligned across a cache line boundary or is in a clean page, the accessed and dirty bits for an incorrect page table/directory entry may be set.

Implication: An operating system that uses hardware task switching may encounter this erratum.

Workaround: The operating system could align all TSSs to be within page boundaries and set the A and D bits for those pages to avoid this erratum. The operating system may alternately use software task management.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C48. Cross Modifying Code Operations on a Jump Instruction May Cause a General Protection Fault

Problem: The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called Cross-Modifying Code (XMC). Software using XMC to modify the offset of an execution transfer instruction (i.e., Jump, Call etc.), without a synchronizing instruction may cause a General Protection Fault (GPF) when the offset splits a cache line boundary.

Implication: Any application creating a (GPF) would be terminated by the operating system.

Workaround: Programmers should use the cross modifying code synchronization algorithm as detailed in Volume 3 of the Intel Architecture Software Developer's Manual, section 7.1.3, in order to avoid this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C49. Deadlock May Occur Due To Illegal-Instruction/Page-Miss Combination

Problem: Intel's 32-bit Instruction Set Architecture (ISA) utilizes most of the available op-code space; however some byte combinations remain undefined and are considered illegal instructions. Intel processors detect the attempted execution of illegal instructions and signal an exception. This exception is handled by the operating system and/or application software.

Under a complex set of internal and external conditions involving illegal instructions, a deadlock may occur within the processor. The necessary conditions for the deadlock involve:

1. The illegal instruction is executed.
2. Two page table walks occur within a narrow timing window coincident with the illegal instruction.

Implication: The illegal instructions involved in this erratum are unusual and invalid byte combinations that are not useful to application software or operating systems. These combinations are not normally generated in the course of software programming, nor are such sequences known by Intel to be generated in commercially available software and tools. Development tools (compilers, assemblers) do not generate this type of code sequence, and will normally flag such a sequence as an error. If this erratum occurs, the processor deadlock condition will occur and result in a system hang. Code execution cannot continue without a system RESET.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C50. FLUSH# Assertion Following STPCLK# May Prevent CPU Clocks From Stopping

Problem: If FLUSH# is asserted after STPCLK# is asserted, the cache flush operation will not occur until after STPCLK# is de-asserted. Furthermore, the pending flush will prevent the processor from entering the Sleep state, since the flush operation must complete prior to the processor entering the Sleep state.

Implication: Following SLP# assertion, processor power dissipation may be higher than expected. Furthermore, if the source to the processor's input bus clock (BCLK) is removed, normally resulting in a transition to the Deep Sleep state, the processor may shutdown improperly. The ensuing attempt to wake up the processor will result in unpredictable behavior and may cause the system to hang.

Workaround: For systems that use the FLUSH# input signal and Deep Sleep state of the processor, ensure that FLUSH# is not asserted while STPCLK# is asserted.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C51. Floating-Point Exception Condition May be Deferred

Problem: A floating-point instruction that causes a pending floating-point exception (ES=1) is normally signaled by the processor on the next waiting floating-point/MMX™ technology instruction. In the following set of circumstances, the exception may be delayed:

1. The excepting floating-point instruction is followed by an instruction that accesses memory across a page (4-Kbyte) boundary, or its access results in the update of a page table dirty/access bit.
2. The memory accessing instruction is immediately followed by a waiting floating-point or MMX technology instruction.
3. The waiting floating-point or MMX technology instruction retires during a one-cycle window that coincides with a sequence of internal events related to instruction cache line eviction.

Implication: The floating-point exception will not be signaled until the next waiting floating-point/MMX technology instruction. This erratum has not been observed in any commercial software applications.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C1AP APIC Access to Cacheable Memory Causes SHUTDOWN

Problem: APIC operations which access memory with any type other than uncacheable (UC) are illegal. If an APIC operation to a memory type other than UC occurs and Machine Check Exceptions (MCEs) are disabled, the processor will enter SHUTDOWN after such an access. If MCEs are enabled, an MCE will occur. However, in this circumstance, a second MCE will be signaled. The second MCE signal will cause the Intel Celeron processor to enter SHUTDOWN.

Implication: Recovery from a PIC access to cacheable memory will not be successful. Software that accesses only UC type memory during APIC operations will not encounter this erratum.

Workaround: Ensure that the memory space to which PIC accesses can be made is marked as type UC (uncacheable) in the memory type range registers (MTRRs) to avoid this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C2AP Write to Mask LVT (Programmed as EXTINT) Will Not Deassert Outstanding Interrupt

Problem: If the APIC subsystem is configured in Virtual Wire Mode implemented through the local APIC (i.e., the 8259 INTR signal is connected to LINT0 and LVT1's interrupt delivery mode field is programmed as EXTINT), a write to LVT1 intended to mask interrupts will not deassert the internal interrupt source if the external LINT0 signal is already asserted. The interrupt will be erroneously posted to the Intel Celeron processor despite the attempt to mask it via the LVT.

Implication: Because of the masking attempt, interrupts may be generated when the system software expects no interrupts to be posted.

Workaround: Software can issue a write to the 8259A interrupt mask register to deassert the LINT0 interrupt level, followed by a read to the controller to ensure that the LINT0 signal has been deasserted. Once this is ensured, software may then issue the write to mask LVT entry 1.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

C3AP Misaligned Locked Access to APIC Space Results in Hang

Problem: When the processor's APIC space is accessed with a misaligned locked access a machine check exception is expected. However, the processor's machine check architecture is unable to handle the misaligned locked access.

Implication: If this erratum occurs the processor will hang. Typical usage models for the APIC address space do not use locked accesses. Systems using such a model will not be affected by this erratum.

Workaround: Ensure that all accesses to APIC space are aligned and/or not locked.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the following documents:

- *Pentium® II Processor Developer's Manual*
- *P6 Family of Processors Hardware Developer's Manual*
- *Intel® Celeron™ Processor* datasheet
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*

All Documentation Changes will be incorporated into a future version of the appropriate Intel Celeron processor documentation.

C1. STPCLK# Pin Definition

The *P6 Family of Processors Hardware Developer's Manual,* the *Pentium® II Processor Developer's Manual* and the *Intel® Celeron™ Processor* datasheet have incorrect definitions of the STPCLK# pin in their alphabetical signal listings. These documents incorrectly state:

> The processor continues to snoop bus transactions and *service interrupts* while in Stop Grant state. When STPCLK# is deasserted, the processor restarts its internal clock to all units and resumes execution.

They should state:

> The processor continues to snoop bus transactions and *may latch interrupts* while in Stop Grant state. When STPCLK# is deasserted, the processor restarts its internal clock to all units, resumes execution, *and services any pending interrupts*.

C2. Invalidating Caches and TLBs

Section 2.6.4 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* incorrectly states:

> The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and TLBs and sends a signal to the external caches indicating that they should be invalidated also.

It should state:

> The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and sends a signal to the external caches indicating that they should be invalidated also.

C3. Handling of Self-Modifying and Cross-Modifying Code

Section 7.1.3 paragraph 1. of the *Intel Architecture Software Developer's Manual Vol 3: System Programming* incorrectly states:

> The act of a processor writing data into a currently executing code segment with the intent of executing that data as code is called **self-modifying code**. Intel Architecture processors exhibit model-specific behavior when executing self-modified code, depending upon how far ahead of the current execution pointer the code has been modified. As processor architectures become more complex and start to speculatively execute code ahead of the retirement point (as in the P6 family processors), the rules regarding which code should execute, pre- or post-modification, become blurred. To write self-modifying code and ensure that it is compliant with current and future Intel Architectures one of the following two coding options **should** be chosen.

It should state:

> The act of a processor writing data into a currently executing code segment with the intent of executing that data as code is called **self-modifying code**. Intel Architecture processors exhibit model-specific behavior when executing self-modified code, depending upon how far ahead of the current execution pointer the code has been modified. As processor architectures become more complex and start to speculatively execute code ahead of the retirement point (as in the P6 family processors), the rules regarding which code should execute, pre- or post-modification, become blurred. To write self-modifying code and ensure that it is compliant with current and future Intel Architectures one of the following two coding options **must** be chosen.

Section 7.1.3 paragraph 6. of the *Intel Architecture Software Developer's Manual Vol 3: System Programming* incorrectly states:

> The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called **cross-modifying code**. As with self-modifying code, Intel Architecture processors exhibit model-specific behavior when executing cross-modifying code, depending upon how far ahead of the executing processors current execution pointer the code has been modified. To write cross-modifying code and insure that it is compliant with current and future Intel Architectures, the following processor synchronization algorithm **should** be implemented.

It should state:

> The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called **cross-modifying code.** As with self-modifying code, Intel Architecture processors exhibit model-specific behavior when executing cross-modifying code, depending upon how far ahead of the executing processors current execution pointer the code has been modified. To write cross-modifying code and insure that it is compliant with current and future Intel Architectures, the following processor synchronization algorithm **must** be implemented.

C4. Machine Check Architecture Initialization of MCi_STATUS Registers

Section 12.5, the last paragraph of the *Intel Architecture Software Developer's Manual Vol. 3: System Programming Guide* incorrectly states:

> The processor can write valid information (such as an ECC error) into the MCi_STATUS registers while it is being powered up. As part of the initialization of the MCE exception handler, software might examine all the MCi_STATUS registers and log the contents of them, then rewrite them all to zeroes. This procedure is not included in the initialization pseudocode in Example 12-1.

It should state:

> The processor can write valid information (such as an ECC error) into the MCi_STATUS registers while it is being powered up. As part of the initialization of the MCE exception handler, software might examine all the MCi_STATUS registers and log the contents of them, then rewrite them all to zeroes. <u>Following power cycling, the MCi_STATUS registers are not guaranteed to have valid data until after the registers are initially cleared to all zeroes by software</u>. This procedure is not included in the initialization pseudocode in Example 12-1.

C5. LOCK# Signal Prefix Operands

Page 3-273, the first sentence of the third paragraph of the *Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference* states:

> The LOCK prefix can be prepended only to the following instructions and to those forms of the instructions that use a memory operand: ADD, ADC, AND, …

It should state:

> The LOCK prefix can be prepended only to the following instructions and to those forms of the instructions that use a memory operand as the destination operand only: ADD, ADC, AND, ...
>
> If the LOCK prefix is used with the memory operand as the source operand then an Invalid Opcode (Undefined Opcode), #UD, can occur.

C6. SMRAM State Save Map Contains Documentation Errors

In the *Intel Architecture Software Developer's Manual Volume 3: System Programming Guide,* revision 2, Chapter 12, "System Management Mode (SMM)," Table 12-1 incorrectly documents the SMBASE+Offset for LDT Base on the P6 family of processors.

> The storage locations for these parameters are model specific (i.e., they may differ between the Pentium® processor, the Pentium® Pro processor, and other P6 family proliferations). **These entries in the tables above will be changed to Reserved. Hardware and software may not rely on the contents of these Reserved regions**.

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the following documents:

- *Pentium® II Processor Developer's Manual*
- *P6 Family of Processors Hardware Developer's Manual*
- *Intel® Celeron™ Processor* datasheet
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*

All Specification Clarifications will be incorporated into a future version of the appropriate Intel Celeron processor documentation.

C1. PWRGOOD Inactive Pulse Width

In Table 16 of the *Intel® Celeron™ Processor* datasheet, footnote 9 should read as follows:

9. When driven inactive or after $V_{CC_{CORE}}$, $V_{CC_{L2}}$, and BCLK become stable. PWRGOOD must remain below V_{IL},max from Table 8 until all the voltage planes meet the voltage tolerance specifications in Table 6 and BCLK has met the BCLK AC specifications in Table 11 for at least 10 clock cycles. PWRGOOD must rise glitch-free and monotonically to 2.5 V.

C2. Floating-Point Opcode Clarification

Section 3.2 of the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference,* provides detailed descriptions of each Intel Architecture instruction. For some instructions, the clarification phrase below needs to either be added to their existing "Comments" section or a "Comments" section needs to be created with the clarification phrase. The phrase is as follows:

> The (**Instruction** shown in the center column of the table below) instruction is actually a combination of two instructions - the FWAIT instruction followed by (**Instruction** shown in the table). If the (**Instruction** shown in the table) instruction should fault in some way (e.g., page fault), the value of EIP that is passed to the fault handler will be equal to the EIP of the first instruction plus one (i.e., the EIP of the second of the pair of instructions). The FWAIT portion of the combined instruction will have completed execution and will typically not be, nor need to be, re-executed after the fault handler is completed.

The following table lists the affected instructions and the location of the clarification phrase:

Instruction Set Reference Section	Opcode	Instruction	Addition	Addition to Page
FCLEX/FNCLEX-Clear Exceptions	9B DB E2	FCLEX	Add "Comments" section with clarification phrase	3-177
FINIT/FNINIT-Initialize Floating-Point Unit	9B DB E3	FINIT	Add clarification phrase to existing "Comments" section	3-204

Instruction Set Reference Section	Opcode	Instruction	Addition	Addition to Page
FSAVE/FNSAVE-Store FPU State	9B DD /6	FSAVE m94/108byte	Add clarification phrase to existing “Comments” section	3-237
FSTCW/FNSTCW-Store Control Word	9B D9 /7	FSTCW m2byte	Add “Comments” section with clarification phrase	3-250
FSTENV/FNSTENV-Store FPU Environment	9B D9 /6	FSTENV m14/28byte	Add “Comments” section with clarification phrase	3-253
FSTSW/FNSTSW-Store Status Word	9B DD /7	FSTSW m2byte	Add “Comments” section with clarification phrase	3-256
	9B DF E0	FSTSW AX		

C3. MTRR Initialization Clarification

The following sentence should be added to the end of the first paragraph of Section 9.12.5 of the Intel Architecture Software Developer’s Manual, Volume 3: System Programming Guide: “The MTRRs must be disabled prior to initialization or modification.”

C4. Non-AGTL+ Output Low Current Clarification

In Table 6 of the *Intel® Celeron™ Processor* datasheet, the note in **bold** should be added:

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	-0.3	0.7	V	
V_{IH}	Input High Voltage	1.7	2.625	V	2.5 V +5% maximum
V_{OL}	Output Low Voltage		0.4	V	2
V_{OH}	Output High Voltage	N/A	2.625	V	All outputs are open-drain to 2.5 V +5%
I_{OL}	Output Low Current	14		mA	**5**
I_L	Leakage Current for Inputs, Outputs, and I/O		±100	µA	3, 4

Notes:

1. Unless otherwise noted, all specifications in this table apply to all Intel® Celeron™ processor frequencies.
2. Parameter measured at 14 mA (for use with TTL inputs).
3. $(0 \le VIN \le 2.5\ V\ +5\%)$.
4. $(0 \le VOUT \le 2.5\ V\ +5\%)$.
5. **Specified as the minimum amount of current that the output buffer must be able to sink. However, VOL_MAX cannot be guaranteed if this specification is exceeded.**

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the following documents:

- *Pentium® II Processor Developer's Manual*
- *P6 Family of Processors Hardware Developer's Manual*
- *Intel® Celeron™ Processor* datasheet
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*

All Specification Changes will be incorporated into a future version of the appropriate Intel Celeron processor documentation.

C1. RESET# Pin Definition

The *P6 Family of Processors Hardware Developer's Manual,* the *Pentium® II Processor Developer's Manual,* and the *Intel® Celeron™ Processor* datasheet have incorrect definitions of the RESET# pin in their alphabetical signal listings. These documents incorrectly state:

> RESET# must remain active for one microsecond for a 'warm' Reset; for a Power-on Reset, RESET# must stay active for at least one millisecond after V_{CCCORE} and CLK have reached their proper specifications.

They should state:

> For a Power-on or "warm" reset, RESET# must stay active for at least one millisecond after V_{CCCORE} and CLK have reached their proper specifications.